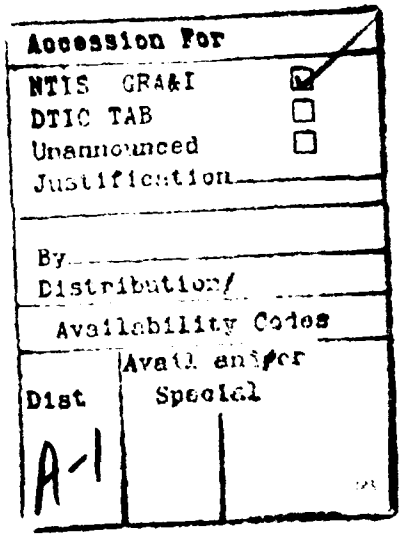
USER'S MANUAL FOR HYPER: A DOMAIN DECOMPOSITION PROGRAM [1]

VERSION 1.0

A. LOUISE PERKINS
INSTITUTE FOR NAVAL OCEANOGRAPHY

[1] This work was performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48 and under the auspices of the Institute for Naval Oceanography.

The Institute for Naval Oceanography (INO) is operated by the University Corporation for Atmospheric Research (UCAR) under sponsorship of the Naval Research Laboratory (NRL). Any opinions, findings, and conclusions or recommendation expressed in this publication are those of the author(s) and do not necessarily reflect the views of NRL.



USER'S MANUAL FOR HYPER: A DOMAIN DECOMPOSITION PROGRAM[1]

Version 1.0

A. Louise Perkins
Institute for Naval Oceanography

[1]This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract No. W–7405-Eng-48 and under the auspices of the Institute for Naval Oceanography.

ACKNOWLEDGEMENTS

I thank the National Magnetic Fusion Energy Computer Center for allowing me the use of their network of Cray's during this code development, and the Institute for Naval Oceanography (INO) for supporting its evolution from a stand-alone program to a set of utility routines.

TABLE OF CONTENTS

USER'S MANUAL FOR HYPER:
A DOMAIN DECOMPOSITION PROGRAM
Version 1.0

1.0 INTRODUCTION

1.1 Purpose

This document describes the details for a set of utilities programs and subroutines that are collectively referred to as HYPER. The document has been prepared for use by either an end-user who wishes to setup a problem, or a programmer who is attempting to modify the code. The author would greatly appreciate all extensions being sent to her at the e–mail address: perkins@ jupiter.ino.ucar.edu.

The utility HYPER is designed to provide an environment for heterogeneous domain decomposition methods. In this user's manual we describe the philosophy behind the code, how to use the code, and the data structures used for implementation. We also briefly discuss both the Cray–2 and SUN computers, and the UNICOS and UNIX operating system used. Our implementation of a mixed directed-undirected graph to represent mesh relationships in a domain decomposition environment combines procedural scheduling with data flow, asynchronous, parallel communication needs, and is an interesting new feature.

Conceptually, the program can be divided into several components (Figure 1). The three major components, integration, domain management, and user interface, belong to distinct areas of computer science: numerical methods, data structures, and man-machine interfaces. The user is expected to provide the numerical methods. The user interface and all needed data structures, however, are completely transparent to the user.

A program entitled HYPER, that this utility is derived from, was originally developed and implemented in parallel on a Cray-2 multiprocessor using the macrotasking facilities available with the operating system used at the National Magnetic Fusion Energy Computer Center (NMFECC) housed at Lawrence Livermore National Laboratory (LLNL). The NMFECC operating system managed the shared main memory and the four processor units (PU) in tandem. Hence, the data structures have been designed for parallel processing, although this utility has not implemented that feature. HYPER was then ported to a UNIX environment and rearranged as a "utility" to provide more flexibility.

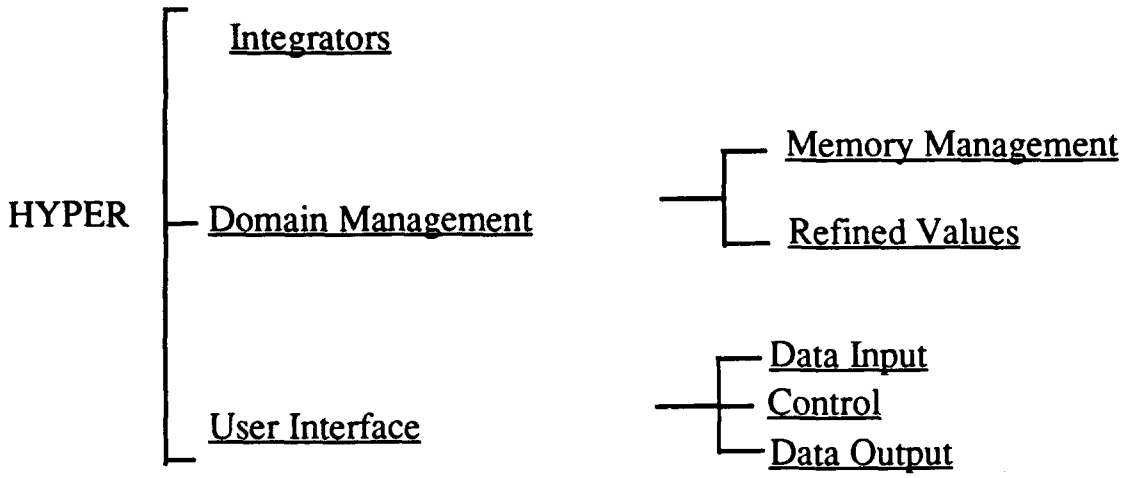


Figure 1. HYPER Components

1.2 Program Design Philosophy

We develop a mixed directed-undirected graph to represent the subdomain relationships. To optimize access speed we implement an inverted index for the directed arcs in the graph.

The code was planned, designed, and debugged using modern software engineering methods. The error handling follows (Aron, 1974). We avoided modularizing where the error handling would become convoluted. Consequently, our error handling needs only information available at the place where the error occurs, allowing local but complete error handling. The program was modularized based on functional cohesion. Each subroutine has a specific task to perform. Because of the original constraint of parallelism, as well as heavy modular communication needs, a shared memory structure is used for global data and variables. It is the responsibility of each subroutine to work within the global environment safely. Toward this end, each parallel subroutine is passed its own subdomain name, S, and processor number, P, along with all grid related indices. This plethora of information defines a separate instance of each grid. A serious stamp coupling occurs when we pass grid relationships between routines. To control such extravagance, utility-provided, abstract-data-type macros are used to access global data structures.

We used the Boehm (1978) metrics for measuring software quality to guide our implementation. The code is portable to the degree that FORTRAN is portable: it is portable in the larger sense that it is self-contained. Due to the modular design, it is reliable, robust, and feasible to extend and modify.

2.0 ACQUIRING THE HYPER CODE

The version of the program described herein is available over Internet via anonymous ftp (user <"anonymous">, password <"anonymous">) from "jupiter.ino.ucar.edu" (IP address 128.160.2.21). Once connected enter "cd/pub/perkins/hyper" to arrive at the appropriate directory. There you will find the following files:

- Readme - Read this first,
- makefile - UNIX system command file,
- hyper.input - Oceanographic data file,
- hyper1.0.f - the model code, and
- hyper1.0.h - the header include file.

If you wish to run the oceanography example, you can acquire the SPEM model via another anonymous ftp and move to the directory pub/perkins/hyper/oceanex. Change to this directory and retrieve all files there. If you have any difficulty retrieving the needed routines remotely, contact:

Dr. Louise Perkins
Building 1103, Room 233
Institute for Naval Oceanography
Stennis Space Center, MS 39529
601/688-3498

3.0 INITIAL CONDITIONS

There are four initial configurations preset in HYPER. These are defined in the subroutine SETIC (SET Initial Conditions), and they also appear in the subroutine CONVCD (CONVergence Check of Data).

The subroutine SETIC has three modes of initializing data: 1) grid aligned partitions; 2) angled partitions; and 3) circular partitions. The user may, of course, provide his/her own SETIC instead. An easier approach would be to provide a function definition, FIC, to SETIC and CONVCD.

The original domain (which is assumed to be rectangular, but could be easily altered) is divided into tiles that are labeled as if they were array elements. For example, the original domain might be divided into four quadrants labeled (1,1), (1,2), (2,1), and (2,2). Tile (1,1) is located in the lower left corner of the rectangle. The second index moves to the right while the first index moves up. Then different values are assigned in each quadrant. There is a maximum of n such

divisions in each of the two spatial directions, x and y. The values for each variable are located in the "tv" arrays. These include:

tvu - velocities in first dimension,
tvv - velocities in second dimension,
tvrho - density, and
tve - energy.

To use the cylindrical setup routine, we partition the domain into the number of disks (NUMDISK) that we wish to tile. There is again a maximum of n tile disks. The disks are filled in with the values from the "tv" arrays. For example, the circle is prescribed as follows: each row is filled up to the radius of the next disk with the appropriate values. Again, the function definition FIC could be used to do this type of initialization. However, the author found this "visual" initialization easier to use.

The angled initial condition is a combination of the array and cylindrical choices. We again subdivide the domain into array-like tiles, but we then fill them in with the circular algorithm, using a constant slope instead of a circle.

3.1 Oceanographic Initial Conditions

There is also a set of initialization routines for an oceanographic simulation. The Semi-spectral Primitive Equation Model (SPEM) developed by Haidvogel, et al (1991) is an example of how to connect your own integration routine into HYPER. The initialization of SPEM is described in a document that must be acquired separately from this document.

4.0 BOUNDARY CONDITIONS

We provide for Neumann (N), Dirichlet (D), Robbins (R), Sponged (S), and user-provided (U) boundary conditions.

The boundary conditions are applied when a routine calls the "func–n" subroutines (a four-sided or "Box" model called func-1, func-2, func-3, and func-4). By convention, one is always the "top" of the "box" and numbering proceeds clockwise.

There is a hierarchy of boundary conditions that are applied. First, Neumann and Robbins boundaries are calculated. Then, any Dirichlet values that need to be interpolated for refined meshes are calculated. Overlapping regions exchange information. Then sponged outflow boundary conditions are calculated.

In summary:

- Neumann and Robbins Boundaries Advanced,
- Interpolate Dirichlet Values onto Refined Meshes,
- Sponge Outflow Boundaries, and
- Shared Boundaries Exchanged.

Because grids can be arbitrarily overlapping and inter-related, determining the location and relative position of grids that overlap is complex. Overlapping grids can be solved implicitly using Schwarz alternating boundary conditions. However, the individual grids also have Neumann and Dirichlet or Robbins boundary conditions. This is handled in the routine NDSS (Neumann Dirichlet Sponged Schwarz). The data structure, IGRDES, keeps the boundary information for all grids.

5.0 THEORY OF OPERATION

The HYPER system operates on a "run" basis. A run consists of a set of initial conditions together with matching boundary conditions, a model or solver task, and the run parameters. The user begins by executing the menu program "HYPER", and choosing the run-time options desired.

5.1 Menus

The following menu discussion explains how the menu system works.

The computer system communicates with the user via menus, reports, and question/answer dialogue. These communication tools are collectively known as the user interface. Most information is entered through question and answer sessions. Confirmation messages appear whenever a transaction is completed.

Defaults

Some questions have options displayed. For example: "Output to the terminal or printer (T/P)." The first option in the list is the default response (e.g., the "T" above).

Menu Traversal

Movement from one menu to another is standardized. Moving through the menu structure is called traversal. The HYPER menu structure is a tree. Each menu in the structure can be thought of as a potential branching point. The main menu is the root of the tree, and presents general categories for options, such as

boundary or initial conditions. Each option can be considered a branching point. As the user climbs the tree, they can select any menu they need to define the problem in any sequence.

To return to a previous menu, enter the option number "0". To exit the menu routine, type "0" in the main menu.

5.1.1 Menu Description

HYPER's menu system begins with the main menu. Here the user is presented with a choice of performing any one of the general categories of functions available. The general categories are described below.

- HYPER Domain Management System
  - Save Definitions for Future Use
  - Run the Program
  - Define Parallel Environment
  - Choose Equations to Solve
  - Define the Grids to Use
  - Define Output Requirements

More specifically, below are the currently available menus.

- HYPER Menu
  - 1 Save
  - 2 Run
  - 3 Parallel
  - 4 Equations
  - 5 Grids
  - 6 Output Options

- SAVE Menu
  - 1 Save Setup Information
  - 2 Add Run Results to Statistical Measures
  - 3 Read Previous Setup Information

- PARALLEL Menu
  - 1 Number of Processors to Use
  - 2 Maximum Asynchronous Lag

- EQUATIONS Menu
  - 1 Known Solution Equations
  - 2 Unknown Solution Equations
  - 3 Choose Discretization
  - 4 Artificial Viscosity Selections

- KNOWN SOLUTION Menu
  - 1 Constant Solution
  - 2 HYPGS
  - 3 2-D Heat Equation
  - 4 Lagrange
  - 5 HYPER

- UNKNOWN SOLUTIONS Menu
  - 1 HYPJAC
  - 2 IGMY
  - 3 BOXMG
  - 4 ADVEC
  - 5 ATF

- SOLUTION Menu
  - 1 Choose Method
  - 2 Define Initial Conditions

- ARTIFICIAL VISCOSITY Menu
  - 1 User Supplied
  - 2 Von Neumann-Richtmyer Type
  - 3 Tensor Type
  - 4 Real Parabolic Term
  - 5 No Artificial Viscosity

- GRID Menu
  - 1 Enter Number of Levels to Refine
  - 2 Do Not Limit Refinement Levels
  - 3 Set Refinement Ratios
  - 4 Domain Decomposition Method Choices

- DOMAIN DECOMPOSITION CRITERIA Menu
  - 1 Local Truncation Error
  - 2 Relative Derivatives (RDD) Method
  - 3 Absolute Derivative Decomposition
  - 4 User Specified

- OUTPUT Menu
  - 1 Reports
  - 2 Graphics

- REPORTS Menu
  - 1 Subroutine Summary
  - 2 Time Summary
  - 3 Options Chosen
  - 4 Parallelism Report

- GRAPHICAL CHOICES Menu
  - 1 Choice of Output Media
  - 2 Two-Dimensional Plots
  - 3 Three-Dimensional Plots
  - 4 Unused
  - 5 Mesh Refinement Plots

5.2 Data Structures

The choice of data structures influences the parallelization, efficiency, and the manageability of a mesh refinement program. We introduce a mixed directed-undirected graph that combines both communication and scheduling needs. An inverted index is maintained for the directed graph to improve code performance and readability.

We introduce a data structure designed to work with a parallel domain decomposition algorithm. We begin with a brief review of domain decomposition. Then, we introduce the domain decomposition algorithm that was used to investigate convection-diffusion equations in Perkins (1989). The mixed directed-undirected data structure is presented. The implementation of the two cooperating data structures involves both a directed inverted graph and a flat-file undirected relation.

We now introduce the motivation for refined meshes. It is well known that to refine the entire coarse mesh in space for mostly smooth fluid dynamics modeled by hyperbolic equations would be inefficient. It would require large amounts of memory and waste processor time in the quiescent regions. In short, refining the entire coarse mesh is overkill. For less smooth fluid dynamics modeled by mixed hyperbolic-parabolic explicit equations, the mesh must also be severely refined in time, resulting in an excessive number of time steps. The small parabolic terms describe the physical behavior within shock, internal, and boundary layers which cannot always be represented by the reduced hyperbolic equation. For example, when modeling shocks in gas dynamics, the Navier-Stokes equations can be reduced to the hyperbolic Euler equations of gas dynamics which

admit discontinuous solutions. However, if the behavior within the shock layer is not known *a priori*, and is of interest, the physical parabolic terms should appear in the numerical model. Scroggs gives an example showing that the reduced hyperbolic equation may place the shock in the wrong location. Alternatively, an implicit solution on a fully refined mesh results in a very large matrix problem. Local mesh refinement can be computationally efficient for these mixed hyperbolic-parabolic equations when the software implementation of the domain decomposition method is efficient.

In this section we present the sequencing for one coarse time step of magnitude $\Delta_c t$ from time t^n to time $t^n + \Delta_c t$, where n indexes the discrete time steps on the coarse mesh. Let the temporal refinement ratio from the coarse mesh to the refined meshes be the integer r, and notate this $r\Delta_f t = \Delta_c t$, so that a subscript "c" informs us that we are on the coarse mesh, while a subscript "f" informs us that we are discussing one of the refined (fine) meshes.

We begin with an overview of the domain decomposition serialized algorithm that describes the procedures we undergo.

Domain Decomposition Serialized Algorithm

```
Advance coarse mesh
Detect overlap of existing subdomains
DO r times
      DO while overlapping fine meshes have not converged
            Iteratively solve implicit equations on refined meshes
            while asynchronously sending boundary information
      ENDWHILE
ENDDO.
Conservatively average refined values onto coarse mesh
```

The corresponding data flow of this algorithm is depicted in Figure 2. The values on the coarse mesh at time $t^n = n\Delta_c t$ are sent concurrently to both the coarse integrator and the refinement criterion. The refinement criterion determines which points on the coarse mesh are above the *a priori* threshold for insignificant parabolic activity. These coarse mesh values may also be conditionally sent to the Lagrange subdomains when we need to interpolate values from the coarse mesh to any newly refined areas. These spatially refined Lagrange meshes exist from time t^n to $t^n + \Delta n_c t$. Each of these refined subdomains is provided with full boundary conditions derived from the coarse mesh, so that they seem "independent". This independence simplifies our mesh-based data structures for the parallel implementation.

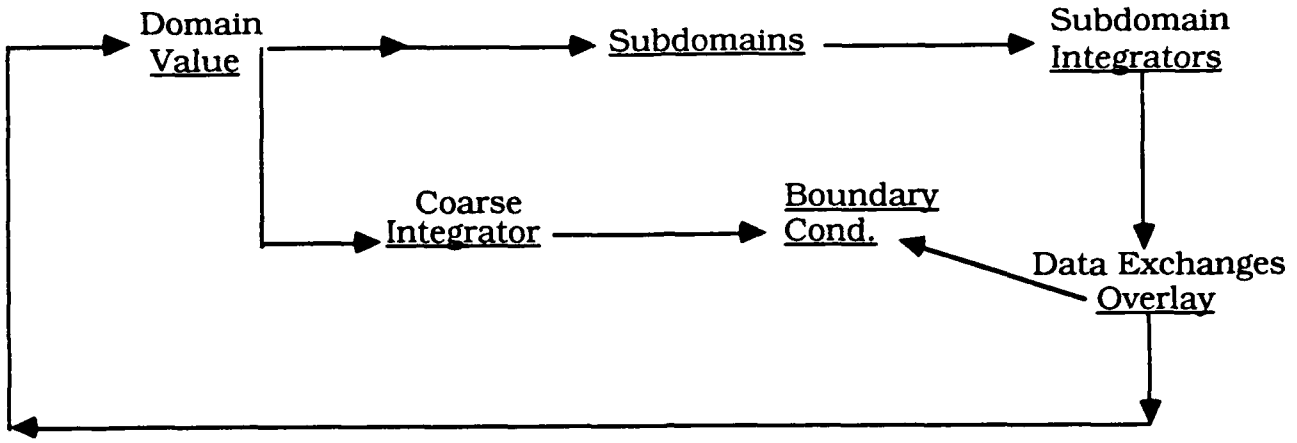


Figure 2. Hyper Data Flow

Of course, overlapping meshes are not independent. Overlapping subdomains must interact to ensure that the necessary data dependencies on connected implicit regions are provided. This is accomplished with asynchronous bi-directional data exchanges.

When all of the refined meshes have been advanced r refined time steps to the next coarse time step, their values at time t^{n+1} are passed to the coarse mesh where a conservative averaging produces an aggregate solution over the entire mesh. Let F^A, F^1, and $\{F^k\}_{k=2}^{p(t)}$ represent the discrete operators for the aggregate solution on the original discretized mesh Ω^1, the solution on the coarse Eulerian mesh Ω^1, and the separate solutions on each of the refined Lagrange subdomains $\{\Omega^k\}_{k=2}^{p(t)}$, respectively. Then the aggregate solution on the entire discretized mesh is given by

$$F^A = C[\{F^k(\Omega^k)\}_{k=2}^{p(t)}\ F^1(\Omega^1)],$$

where the operator C projects from the refined grid onto the coarse grid using the F^k where available and F^1 elsewhere.

The choice of data structures for mesh refinement is important as stressed in Bolstad (1982) and as illustrated in Berger (1983). The data structures used influenced the parallelization, efficiency, and ease of implementing an algorithm. Although a tree structure is most popular for multidimensional adaptive mesh programs, it proved too restrictive for the parallel implementation. The data structure presented here is general purpose and could be applied to other parallel solution methods whose serial algorithm was tree based and where the siblings must interact. It couples procedural scheduling constraints with asynchronous communication needs by representing the topological structure of mesh relationships.

In previous work the data structures used to implement similar methods range from simple lists to linked lists to tree structures. Bolstad (1982) discusses linked lists for domain decomposition in detail while Berger (1983) presents tree structures.

5.3 Mixed Directed-Undirected Graph

Each of the available processors is given a subproblem, $F^k = 0$, defined on the subdomain Ω^k. The mesh configuration is static between the coarse-refined time step barriers. Consider this static configuration as a graph with both directed and undirected edges, $G = (V, E_D, E_U)$, where V is the set of all meshes, E_D are directed edges that express the coarse-refined relationship, and E_U are undirected edges that indicate overlapping refined meshes. This structure admits the possibility of unbounded levels of refinement and multiple coarse meshes. Although our implementation has only one coarse mesh and one level of refinement, we implemented a more general domain decomposition data structure for flexibility in future applications. The local uniform mesh refinement method of Berger (1983), which requires multiple levels of refinement, and the multiple, specialized body conforming meshes used at NASA for different portions of an aircraft are both interesting domain decomposition techniques. We did not want to exclude these alternate approaches from our domain decomposition data structure.

An example of a mixed directed-undirected graph is given in Figure 3. For coarse mesh 1 with refined meshes 2, 3, and 4, mesh 2 is further refined into meshes 5, 6, and 7; and mesh 4 is refined again using meshes 8 and 9. This information is needed for procedural scheduling and is stored in the directed-graph structure which is implemented using an inverted index. In the figure, the directed-graph inverted index structure has two columns. The first column is the ordered index of existing meshes, and the second column is a pointer to each existing meshes original parent.

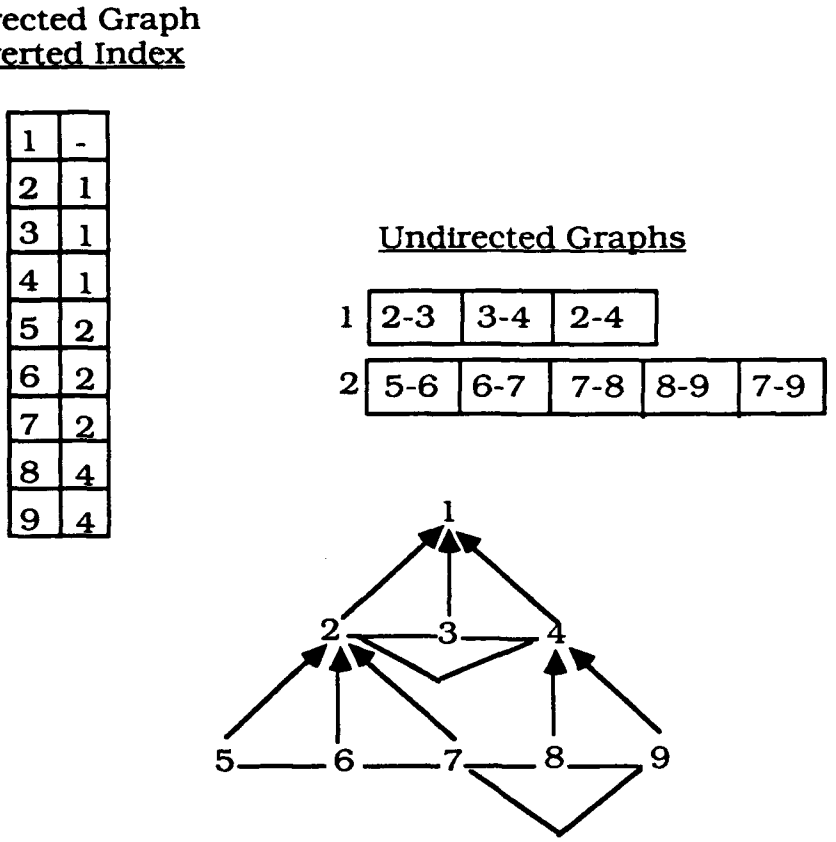


Figure 3. Mixed Directed-Undirected Graph

At the second level of the mixed directed-undirected graph, we see that meshes 2 and 3, 3 and 4, and 2 and 4 overlap. This is illustrated with lines connecting these meshes in Figure 3. These relationships are contained in the undirected graph marked 1. At the third level we see that meshes 5 and 6, 6 and 7, 7 and 8, 8 and 9, and 7 and 9 overlap. These relationships are contained in the undirected graph marked 2. To our knowledge, this data structure has not been used explicitly for finite difference domain decomposition solutions before. Bell (1983) reviewed large-scale scientific computing data structures. Concluding that the nature of scientific computing constrains the choice of data structures, she represents these constraints using a relational model of data--a graph. Also, the only data structures that she found in existing codes were arrays, lists, and trees. That is, although she chose to represent the data found in her survey as a graph, no actual implementation represented it that way. This is because general graphs can be expensive to build, maintain, and access. To overcome these drawbacks, we combined the ease of accessing and managing an indexed tree with the generality of a graph. A tree structure is used whenever possible. When not possible, and in our implementation, this corresponds to overlapping refined meshes, which are represented by the undirected arcs. A flat-file relation for the graph is utilized. This scheme is feasible when the size of the individual disjoint graphs is not large.

The directed arcs, G_D, are used to ensure that the underlying coarser meshes have advanced and filled in the boundary conditions for the overlaid refined meshes, as required for the Lagrangian advancement. They represent the procedural connections between meshes. The undirected arcs, G_U, are used to coordinate the asynchronous iterations on the overlapping meshes. They represent cooperating but independent meshes, and they determine the asynchronous data flow. The combined mixed directed-undirected graph is used by a load-balancing algorithm to schedule the subproblems on to the available processors (Perkins, 1989).

To efficiently manage the parallel advancement of the independent refined Lagrange meshes, we relinquish coarse-mesh supervision of the refined meshes, freeing them to execute their own tasks within parameters established by the underlying coarse mesh at the previous coarse-refined time step barrier. The coarse mesh creates each of its overlaid refined meshes; provides them with their needed boundary conditions and their connectivity graph, which is one disjoint graph from the mixed directed-undirected graph G; and tells the refined mesh the number of time steps to the coarse-refined synchronization barrier. The coarse mesh knows how many overlaid refined meshes were created, and it waits until all of them have contacted it with an "all done" message packet containing the overlaid values. Each refined mesh knows how many time steps to take, and with which other refined meshes it must exchange data.

The parallel scheduler uses the disjoint graph structure to schedule spatially close refined meshes onto topologically close processors. Figure 4 shows the Euler-Lagrange synchronization graph for a three processor implementation. The coarse mesh (DD^i) is easily advanced in parallel on all available processors, without considering the topological structure of the separate grids provided the grid overlap always contains the domain of dependence needed by an explicit solver.

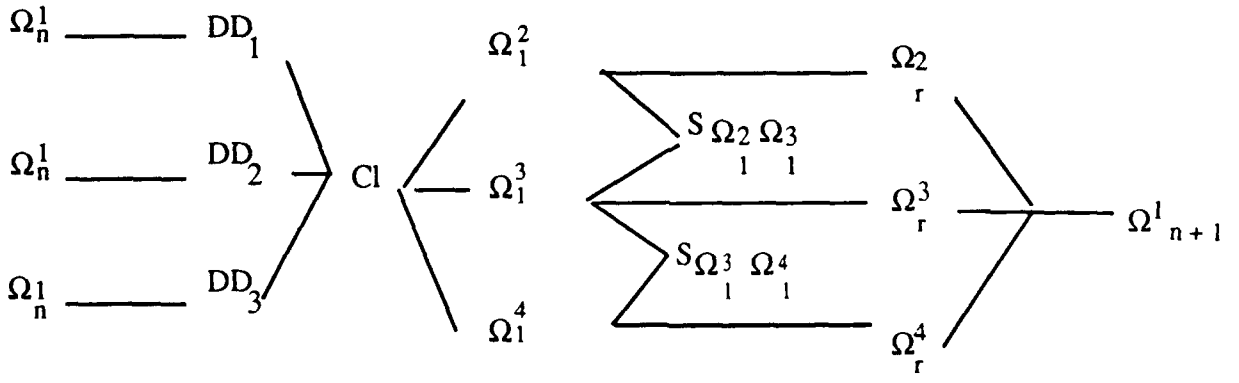


Figure 4. Euler-Lagrange Synchronization Graph

Clustering can aiso proceed in parallel on the same coarse mesh subdivisions. In our implementation, for the problems examined in Perkins (1989), the parallel clustering was efficient. However, our parallel clustering algorithm can theoretically degenerate to a worse-case scenario where nothing is gained over a serial implementation.

After all parallel clustering routines have completed, we have a global synchronization cluster barrier, *CL*. This can be relaxed, but not completely eliminated, as our worse-case scenario indicates.

Each refined mesh is then assigned to an available processor in this example. As mentioned before, the undirected arcs are used to indicate refined mesh dependencies. This information is used to assign processors in a sympathetic configuration. In our example here all three refined meshes overlap, Ω^2 with Ω^3, and Ω^3 and Ω^4. On a linear array of processors, connected p1-p2-pn, we would assign Ω^{n+1} to processor *pn*. Each overlapping array must undergo asynchronous, bi-directional data exchanges. This activity is indicated with $S\Omega_i^k\Omega_i^j$ in the figure.

When all refined meshes complete r Δ_f time steps, the coarse-refined synchronization barrier is crossed, and Ω_{n+1}^1 has been calculated.

5.4 Inverted Graph

The directed graph, G_D, is a reverse-linked tree that points from child-to-parent as previously shown in Figure 3. Parent-to-child tree structures are frequently used to monitor mesh refinement (Berger, 1983). However, it is the child (refined mesh) which must contact the parent (coarser mesh) when it has finished execution in a parallel implementation. In our implementation of the directed graph, we extract these frequently used traversal paths from the data and keep them in an index. This allows the majority of G_D traversals to occur quite rapidly. It also provides easy management of "adoption" and "joint custody."

The inverted index tree is functionally equivalent to a classically linked tree structure up to the order of the siblings.

An example of an extracted traversal index for G_D is given in Figure 5. At the creation of each refined mesh, the underlying coarse mesh knows the identifications of all of its refined meshes. It keeps this information for each of its refined meshes. In the figure, mesh 1 knows its children are meshes 2, 3, and 4, while mesh 4 knows it has two children, meshes 8 and 9. As each first level refined mesh 2, 3, or 4 reaches its respective coarse-refined synchronization barrier, it waits for messages to be sent from each of its refined mesh children; parent-to-

child paths are not needed. As each refined mesh reaches the coarse-refined synchronization barrier, it puts together a message packet of its advanced values and looks up who its parent meshes are. It then sends the packet to the underlying coarser parent meshes. Child-parent communication accounts for the most frequent access to G_D, so the physical structure of the inverted index reflects this.

Index

Node Num.	Ptr. Parent
1	0
2	1
3	1
4	1
5	2

Logical Tree

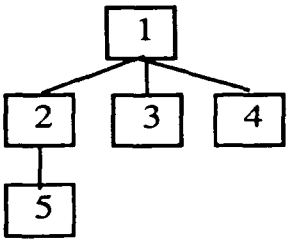


Figure 5. Inverted Tree

5.5 Ease of Use

Because our refined meshes move across the coarse mesh, refined meshes can migrate from one coarser mesh to another. The logical tree reflects the nesting arrangement at a specific instance in time. If a refined mesh moves from one parent mesh to another, then the child mesh (who knows its original parent) sends a message to the previous parent, allowing it to cross their joint coarse-refined synchronization barrier without receiving any data from the refined mesh that has migrated, then looks up its new parent in the inverted tree, and sends its advanced values to the new parent, the migrating child mesh.

Refined meshes can also be shared by more than one parent as they move across the coarser mesh. This underlying structure is, then, not strictly a tree. However, no change is required to our inverted index structure except to allow more than one value for the parent pointer. Also, no synchronization problems arise for the adopted parent due to this migration. The child contacts its previous parent and informs it that it is migrating to a new parent. The new parent is also contacted and informed that it has successfully adopted a new child mesh.

For information that is specific to each mesh, this mesh number points to the information stored for that subdomain, such as the time step size, the number of time steps advanced so far for this mesh, the time the mesh was created, the number of data points in each dimension, the step size in each dimension, a pointer to its location within the parent's data arrays, the level of the mesh's refinement, its underlying coarse mesh, and the boundary conditions to use for that mesh.

5.6 Flatfile Undirected Relations

The undirected graph G_U represents sovereign relationships; no procedural control distinctions are made between any two meshes connected by an undirected path. Either mesh can initiate communication or decide when to advance across the refined-refined synchronization barrier. We use these paths to represent the bi-directional iterations (Schwarz, 1890) where either mesh can decide when to initiate an iteration and when to advance across the refined-refined synchronization barrier.

These undirected graphs are implemented in a flat-file relation. This is illustrated in Figure 6. In a list, labeled "Forest of Disjoint Graphs" in the figure, we keep a contiguous record of each disjoint graph. The length of the graph, which includes the preface length field, is first. It is then followed by the graph which is stored in the form shown in the table "Graph Relations". This is a two-tuple of three-tuples. One three–tuple mesh relationship is used. It contains the number of the mesh, the level (or depth) of the mesh within the directed tree, and a pointer to the parent mesh. There is one two-tuple used each time any two meshes overlap. In the figure there are two disjoint subgraphs, $Gr(1)$ and $Gr(2)$. $Gr(1)$ knows that meshes 4 and 5 overlap, and $Gr(2)$ knows that meshes 7 and 8 overlap.

The mixed directed-undirected graph allows us to combine procedural scheduling needs with asynchronous communication needs. Because the directed graph is inverted it provides for liberal mesh relationships such as migration in an efficient way. The combined structure allows us to present both the procedural scheduling and the asynchronous communication needs in an integrated format to a parallel scheduling routine. The scheduler can then quickly decide which processors to use for which subproblems using heuristic algorithms. That is, we provide our scheduler with information on data dependence together with our scheduling constraints.

This new data structure provides efficient, readable software for domain decomposition algorithms in a parallel environment.

Total Length = $\sum_{i=1}^{n}$ Len(i)

Forest of Disjoint Graphs

Len(1)	graph	Len(n)	graph	λ
7	Gr(1)	7	Gr(2)	-1

Graph Relations

	num	lvl	prt	num	lvl	prt
Gr(1)	4	3	2	5	3	2
Gr(2)	7	3	3	8	3	3

Figure 6. Graph Structures

REFERENCES

Aron, J.D. (1974). The Program Development Process-The Individual Programmer, *Addison-Wesley*, 162-164.

Babuska, I., J. Chandra, and J.E. Flaherty (1983). Adaptive Computational Methods for Partial Differential Equations, *SIAM*, Philadelphia, PA, (0 89871 191 6).

Bell, J.L. (1983). Data Structures For Scientific Simulation Programs, *Ph.D. Thesis*, University of Colorado.

Berger, M.J. (1982). Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations, *Ph.D. Dissertation*, Stanford University.

Berger, M.J. (1983). Data Structures for Adaptive Mesh Refinement. *In: Adaptive Computational Methods for Partial Differential Equations, SIAM,* Philadelphia, PA (0 89871 191 6).

Berger, M.J. and P. Colella. Local Adaptive Mesh Refinement for Shock Hydrodynamics, to appear in *J. Comput. Phys.*, *82*, 64-84.

Boehm, B.W., J.R. Brown, and M. Lipow (1978). Quantitative Evaluation of Software Quality, *Proceedings, 2nd International Conference on Software Engineering*, IEEE Computer Society, Long Beach CA (78 67758), 286-299.

Bolstad, J.H. (1982). An Adaptive Finite Difference Method for Hyperbolic Systems in One Space Dimension, Lawrence Berkeley Lab Report LBL 13287.

Chin, R.C.Y., G.W. Hedstrom, J.R. McGraw, and F.A. Howes (1985). Parallel Computation of Multiple-Scale Problems, Lawrence Livermore National Laboratory Report UCRL-92007, Rev. 1.

Haidvogel, D., J. Wilkin, and R. Young (1989). A Semi-Spectral Primitive Equation Ocean Circulation Model Using Vertical Sigma and Orthogonal Curvilinear Horizontal Coordinates, *J. Comput. Phys.*, *94*, 151-185.

Hedstrom, Kate (1990). User's Manual for a Semi-Spectral Primitive Equation Regional Ocean-circulation Model Version 3.0ß, *Institute for Naval Oceanography, SP-1*, 82 pp.

Horowitz, E.J. (1987). QN3D: A Three Dimensional Quasi Neutral Hybrid Particle in Cell Code with Applications to the Tilt Mode Instability in Field Reversed Configurations, *Ph.D. Thesis*, UC Davis Report UCRL-53808.

ISSC (1985). DISSPLA Display Integrated Software System and Plotting Language User's Manual, Version 10.0, San Diego, CA.

Leibovich, S. and A.R. Seebass (1974). Nonlinear Waves, Cornell University Press, Ithaca, NY.

Meisel, W.S. (1972). Computer-Oriented Approaches to Pattern Recognition, Academic Press, 145-146.

Oran, E.S. and J.P. Boris (1987). Numerical Simulation of Reactive Flow, Elsevier Science Publishing Co., New York (0-444-01251-6).

Perkins, A.L. (1988). Tailoring Domain Decomposition to the Network Structure for Parallel Processing Of Fluid Dynamics, Lawrence Livermore National Laboratory Report UCID-21609.

Perkins, A.L. (1989). Parallel Heterogeneous Mesh Refinement For Multi-dimensional Convection-Diffusion Equations Using an Euler-Lagrange Method, *Ph.D. Thesis*, UCRL-53950.

Schlichting, H. (1979). Boundary Layer Theory, McGraw Hill Book Company, New York, NY, 47-69 (0 07 055334 3).

Schwarz, H.A. (1890). Ueber Einen Grenzübergang Durch Alternirendes Verfahren, *Gesammelte Mathematische Abhandlungen 2*, Springer Verlag, Berlin.

Scroggs, J.S. (1988). The Solution of a Parabolic Partial Differential Equation via Domain Decomposition: The Synthesis of Asymptotic and Numerical Analysis, *Ph.D. Thesis*, University of Illinois.

Smoller, J. (1983). Shock Waves and Reaction-Diffusion Equations, Springer Verlag, 255 (0 387 90752 1).

Thompson, J.F., Z.U.A. Warsi, and C.W. Mastin (1985). Numerical Grid Generation Foundations and Applications, North Holland, Amsterdam, 367–369 (0 444 00985 X).

APPENDIX A

Grid Attributes

Each subdomain has a mesh number that is a "pointer" to the information kept about that subdomain, such as the time step size, the number of time steps advanced so far for this mesh, the time the mesh was created, the number of data points in each dimension, the step size in each dimension, a pointer to its location within the parent's data arrays, the level of the mesh's refinement, its underlying coarse mesh, and the boundary conditions to use for that mesh.

The data values are kept in a two dimensional heterogeneous array currently defined in the include file "grids.inc". When a subdomain is needed, this grid is activated and the needed initial conditions and boundary values are filled in. The resulting refined mesh is then computationally independent of its underlying coarse mesh up to its next synchronization barrier.

APPENDIX B

Data Structures

The following are all the data structures used, their layouts, and their owners.

Table 1. Data Structure Overview, Part I

STRUCTURE	DIMENSION	PURPOSE	OWNER
igrdes	(35,ngrids)	Grid Attribute List	
isbgc		sibling graph for overlap	ADV
mngmt	1 - width 2 - length 3 - 4 -	grid management array	ALLOC, DALL
igra		Directed Graph Relations	ADV
gn	(1-20)	grid number	
iw		Load Balancing List	DECOVL
newgin		grid to add	P
iext	0 1	positional array not strict extension strict extension	
itcrr(2,2)	(1,1) (2,1) (1,2) (2,2)	inner iterations for lvl 1 inner iterations for lvl 2 average over "n" items average	
nnewg		list of existing grids	DECG
Map		position in NEWG	G
mrgey	(20)	Closest Grid Overlap	
iptr	(20,20) (i,j)=0 1	relates each grid no overlap overlap	

Table 2. Data Structure Overview, Part II

newg	(10,20)	grids
	1	loc:ulc
	2	loc:llc
	3	loc:ulr
	4	loc:urr
	5	width
	6	length
	7	level
	8	parent pointer
	9	mark field
	10	mark field
sec	(100,6)	timing values
	1-100	routine names
	1	running total
	2	total parallel time
	3	
	4	total parallel time
	5	maximum parallel time
	6	count of parallel calls
icntr	(100)	times routine executed

Details

The igrdes array keeps the information on all currently existing grids. There are currently a maximum of *ngrids* grids, but this can be altered by changing the parameter statement in the include file.

APPENDIX C

Parallelization

A list of all routines scheduled with strtask and waittask (the parallel start and synchronize routines on the Crays) follows. The STUPIF flag indicates whether the serial or parallel call will be invoked.

Table 3. Parallel Scheduling Overview

Routine	Parallel Task
adv	solvrn
clus	grads
rdds	rdd
padvn	adv
ppadv	padvn
xcute	ppadv
solvrn	solvr
grads	grad

APPENDIX D

File Interfaces

This is a list of all files used and their purpose.

Table 4. File Structure and Usage

Owner Routine	File	Structure
Major	timip	1 - used 0 - unused A(9) - identifier
	errf (input)	number of entries message disposition: 0 - report 1 - report and correct 2 - delete
	menf	number of menus list of menus: length of menu in order, menu name in-order means

APPENDIX E

Error Handling

The subroutine error handles all error processing. It has two states: initialize and process. When parameter INIT is not zero, the error messages are read in from file ERRF. This file has format:

- number of messages (I3)
- message (one line max)
- disposition of message

where:

Table 5. Error Handling Summary

Disposition Flag	Action
0	warning
1	error, but caller will handle it
2	error, terminate immediately

DISTRIBUTION LIST

1. Office of Naval Research
   Code 1242 (10 copies)
   800 North Quincy Street
   Arlington, VA 22217-5000

2. Director, Atmospheric Sciences Directorate
   NRL West (Code 400)
   Monterey, CA 93943-5000

3. Commanding Officer
   Fleet Numerical Oceanography Office
   Monterey, CA 93943-5000

4. Commanding Officer
   Naval Oceanographic Office
   Stennis Space Center, MS 39529

5. Technical Director
   CNOC (Code OOT)
   Stennis Space Center, MS 39529

6. Officer In Charge
   NRL (Code 100)
   Stennis Space Center, MS 39529

7. Technical Director
   NRL (Code 110)
   Stennis Space Center, MS 39529

8. Director, Ocean Sciences Directorate
   NRL (Code 300)
   Stennis Space Center, MS 39529

9. Dr. A. D. Kirwan, Jr.
   College of Sciences
   Dept. of Oceanography
   Old Dominion University
   Norfolk, VA 23529-0276

10. Head, Ocean Sensing and Prediction Division
    NRL (Code 320)
    Stennis Space Center, MS 39529

11. Library (3 copies)
    NRL (Code 125)
    Stennis Space Center, MS 39529

12. Dr. William Holland
    National Center for Atmospheric Research
    P.O. Box 3000
    Boulder, CO 80307

13. UCAR Library
    P.O. Box 3000
    Boulder, CO 80307

14. Dr. Albert W. Green, Jr.
    NRL, Code 330
    Building 1105
    Stennis Space Center, MS 39529

15. Professor George L. Mellor
    Princeton University
    P.O. Box CN710, Sayre Hall
    Princeton, NJ 08544-0710

16. Dr. John R. Apel
    Applied Physics Laboratory
    Johns Hopkins University
    Laurel, MD 20723

17. Dr. J. Dana Thompson
    NRL, Code 320
    Stennis Space Center, MS 39529

18. Dr. William J. Schmitz
Dept. of Physical Oceanography
Woods Hole Oceanographic
Institution
Woods Hole, MA 02543

19. Dr. Edward L. Barker
NRL, Code 440
Monterey, CA 93943-5006

20. Professor Otis B. Brown
Division of Meteorology &
Physical Oceanography
RSMAS, University of Miami
4600 Rickenbacker Causeway
Miami, FL 33149

21. Dr. Richard W. Miksad, Chairman
Dept. of Aerospace Engineering
and Engineering Mechanics
The University of Texas at Austin
Austin, TX 78712-1085

22. Professor Allan R. Robinson
Center for Earth &
Planetary Physics
Harvard University, Pierce Hall
29 Oxford Street, Room 100D
Cambridge, MA 02138

23. Dr. Ron McPherson, Director
National Meteorological Center
World Weather Building
5200 Auth Road
Camp Springs, MD 20746

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

gency Use Only *(Leave blank).*	2. Report Date.	3. Report Type and Dates Covered.
	March 1992	Special Report

itle and Subtitle.

Jser's Manual for HYPER: A Domain Decomposition Program
/ersion 1.0

5. Funding Numbers.

Program Element No. 61153N
Project No. R310300
Task No. 801
Accession No. DN250022

uthor(s).

A. Louise Perkins

erforming Organization Name(s) and Address(es).

Institute for Naval Oceanography
Building 1103, Room 233
Stennis Space Center, MS 39529-5005

8. Performing Organization Report Number.

SP-4

Sponsoring/Monitoring Agency Name(s) and Address(es).

Naval Research Laboratory
Stennis Space Center, MS 39529-5004

10. Sponsoring/Monitoring Agency Report Number.

Supplementary Notes.

a. Distribution/Availability Statement.

Approved for public release; distribution is unlimited.

12b. Distribution Code.

Abstract *(Maximum 200 words).*

This document describes the details for a set of utilities programs and subroutines that are collectively referred to as HYPER. The utility HYPER is designed to provide an environment for heterogeneous domain decomposition methods. In this user's manual we describe the philosophy behind the code, how to use the code, and the data structures used for implementation.

Subject Terms.

(U) INO (U) NAOPS (U) SPEM (U) PRINCETON (U) NOGUFS
(U) MODEL (U) PREDICTION (U) HARVARD (U) DART (U) HYPER

15. Number of Pages.

16. Price Code.

Security Classification of Report.	18. Security Classification of This Page.	19. Security Classification of Abstract.	20. Limitation of Abstract.
Unclassified	Unclassified	Unclassified	SAR